# The OCRopus Open Source OCR System

Thomas M. Breuel
DFKI and U. Kaiserslautern
Kaiserslautern, Germany
tmb@iupr.dfki.de

## ABSTRACT

OCRopus is a new, open source OCR system emphasizing modularity, easy extensibility, and reuse, aimed at both the research community and large scale commercial document conversions. This paper describes the current status of the system, its general architecture, as well as the major algorithms currently being used for layout analysis and text line recognition.

## 1. INTRODUCTION

There has been a resurgence of interest in optical character recognition (OCR) in recent years, driven by a number of factors. Search engines have raised the expectation of universal access to information on-line, and cheap networking and storage have made it technically and economically feasible to scan and store the books, newspapers, journals, and other printed materials of the world.

Commercial OCR engines have traditionally been optimized for desktop use–scanning of letters, memos, and other end-user documents; some other engines have been optimized for special applications like bill scanning. However, OCR engines for large-scale digital library applications differ in their requirements from such traditional OCR systems. In addition, OCR systems traditionally have usually been developed for specific scripts and languages. These issues limit the usefulness of such existing OCR systems for large scale digital library applications.

The goal of the OCRopus OCR system is to overcome these limitations.

- OCRopus is an open source OCR system allowing easy evaluation and reuse of the OCR components by both researchers and companies.
- The particular open source license used by OCRopus, the Apache 2 license, simplifies collaboration between commercial and academic researchers, since contributions can be used commercially with few restrictions.
- The system is designed from the ground up with multi-lingual and multi-script recognition; for example, it uses Unicode throughout and relies on the HTML[1] and CSS[2] standards for the representation of typographic phenomena in a wide variety of languages and scripts.
- The system relies on only a small number of intermediate representations and interfaces, most of them image based,[3] making it easy to integrate both existing and new algorithms.
- The system is extensible and programmable in a built-in scripting language.

The rest of this paper will provide an overview of the methods used in OCRopus, as well as some other information of interest to potential users or contributors to OCRopus. Please note that this paper is not a review of OCR; for example, there are many worthwhile and competitive algorithms for layout analysis and character recognition, but this paper will focus on algorithms that are actually used within the OCRopus system.

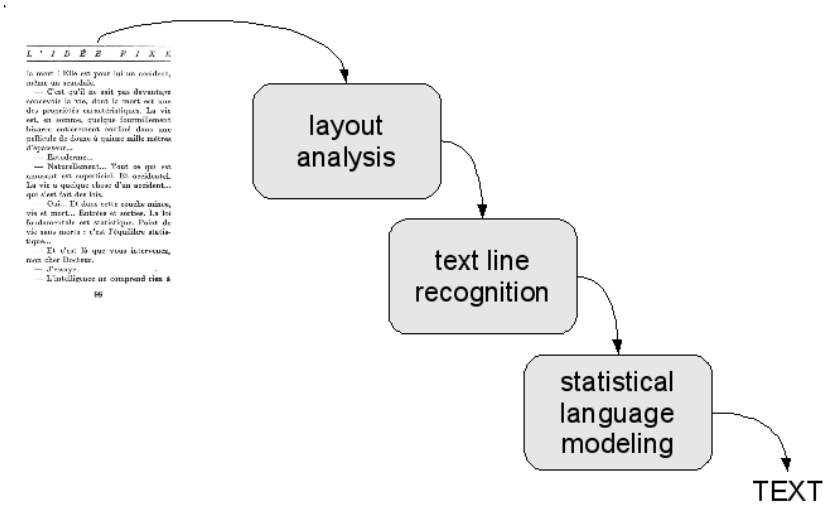


Figure 1. A rough flow diagram of the OCRopus system. The system is strictly feed-forward and consists of three major components: layout analysis, text line recognition, and statistical language modeling. (In addition, it also contains tools for preprocessing and classifier combination.)

## 2. ARCHITECTURE

The overall architecture of the OCRopus OCR system is a strictly feed-forward architecture (no backtracking) with three major components: (physical) layout analysis, text line recognition, and statistical language modeling; it is similar to a previous handwriting recognition system.[4] The individual steps are (Figure 1):

- Physical layout analysis is responsible for identifying text columns, text blocks, text lines, and reading order.
- Text line recognition is responsible for recognizing the text contained within each line (note that lines can be vertical or right-to-left) and representing possible recognition alternatives as a hypothesis graph.
- Statistical language modeling integrates alternative recognition hypotheses with prior knowledge about language, vocabulary, grammar, and the domain of the document.

Text line recognition itself either relies on black-box text line recognizers, including pre-existing ones, or by recognition using over-segmentation and construction of a hypothesis graph (below). An important aspect of the OCRopus system is that we attempt to approximate well-defined statistical measures at each processing step. For example, the default layout analysis component is based on maximum likelihood geometric matching under a robust Gaussian error model. And the MLP-based character recognition, followed by statistical language modeling, approximates posterior probabilities and segmentation probabilities based on training data and then attempts to find the Bayes-optimal interpretation of the input image as a string, given prior knowledge encoded in statistical language models.

## 3. PROCESSING STEPS

Above, we saw generally how the processing steps of the OCRopus system fit together. Let us now look at each of the processing steps in more detail.

$$
\begin{aligned}
& P(W, S|x) \\
&= \frac{P(x|W,S)\,P(W,S)}{P(x)} \\
&\approx P(W) \prod_i \frac{P(x_i|w_i,s_i)\,P(s_i|w_i)}{P(x_i)} \\
&= P(W) \prod_i \frac{P(w_i,s_i|x_i)}{P(w_i)}
\end{aligned}
$$

Figure 2. The Bayesian foundations of the OCRopus OCR system. The original approach of combining a discriminative classifier that estimates posterior probabilities with a segmenter was developed for speech recognition.[5] It has been adapted to handwriting recognition[4] and OCR.

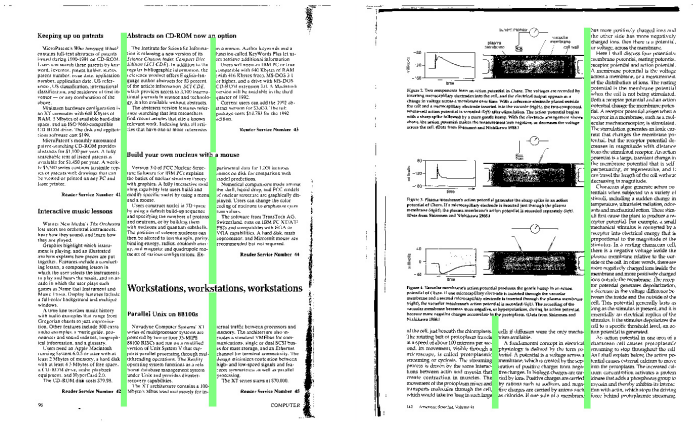

Figure 3. Example of column finding. The whitespace between the columns is identified as maximum area whitespace rectangles with a high aspect ratio and large numbers of adjacent, character-sized connected components.

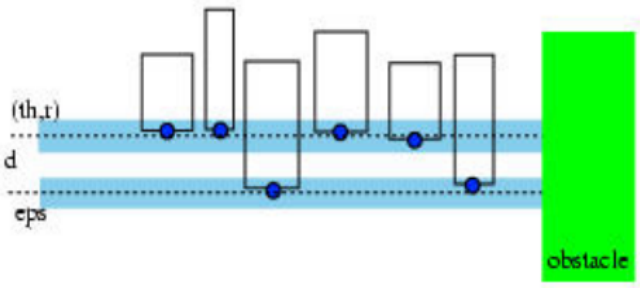


Figure 4. The currently best performing text line extractor in OCRopus searches for globally optimal matches to a precise text line model under a parameterized error model.

| Algorithm | Error rates [%] | | |
|---|---|---|---|
| | Train | Test | |
| | Mean | Mean | Stdev |
| X-Y cut | 14.7 | 17.1 | 24.4 |
| Smearing | 13.4 | 14.2 | 23.0 |
| Whitespace | 9.1 | 9.8 | 18.3 |
| Text-line | 5.6 | 7.0 | 13.3 |
| Docstrum | 4.3 | 6.0 | 15.2 |
| Voronoi | 4.7 | 5.5 | 12.3 |
| **Whitespace-cuts** | **1.7** | **4.4** | **11.1** |

Figure 5. Error rates of the current layout analysis system when applied to the documents in the UW3 database. Error rates are compared to the performance of other, standard methods. (see[11] for more information about experimental conditions.)

### 3.1 Preprocessing and Cleanup

Preprocessing (binarization, image cleanup, skew correction, etc.) is an important part of OCR systems; good preprocessing can greatly improve overall OCR error rates. However, preprocessing is often quite dependent on the application domain and the image capture methods used. In the long term, we would like to automate image preprocessing and cleanup as much as possible (e.g., using a generate-and-test method similar to the one described in[4]). However, in the short term, we provide a scriptable toolbox that permits users to rapidly construct preprocessing and image cleanup pipelines for their specific needs.

OCRopus provides a standard toolbox of binary and grayscale image processing and mathematical morphology routines, all easily invocable from the scripting interface. In addition, OCRopus provides four tools that are less commonly found in other systems:

- A run-length based binary morphology toolbox permitting fast operations using large masks and using non-rectangular masks[6]
- An efficient local adaptive thresholding algorithm based on integral images[7]
- High-accuracy RAST-based skew detection and correction[8]
- Page frame detection,[9] permitting noise in non-content areas to be cropped away and removed

### 3.2 Layout Analysis

The overall goal of layout analysis is to take the raw input image and divide it into non-text regions and "text lines"–subimages of the original page image that each contain a linear arrangement of symbols in the target language. Text lines need not be horizontal, left-to-right; the system imposes no constraints on the shape or direction of the text lines generated by the layout analysis. However, layout analysis modules must indicate the correct reading order for the collection of text lines, and the text line recognizer needs to be able to cope with the direction and nature of the text lines returned by page layout analysis.

#### 3.2.1 Text-Image Segmentation

OCRopus contains a simple text-image segmentation system.[10] It operates by first dividing the input image into candidate regions. Then, features are extracted for each candidate region. Finally, each region is classified using logistic regression into text, grayscale image, line drawing, ruling, and other kinds of regions.

#### 3.2.2 RAST-Based Layout Analysis

The primary layout analysis method used by OCRopus is currently based on two related algorithms, one for whitespace identification, and the other for constrained text line finding.[12] Both methods operate on bounding boxes computed for the connected components of the scanned input page image.

Figure 6. A dynamic programming algorithm generates character segmentation hypotheses[19] (here illustrated for the case of handprinted letters); the algorithm does not require characters to be segmentable using a straight line and hence works in the presence of kerning and italics.

**Column Finding** In the first step, the column finder uses a maximal whitespace rectangle algorithm[12] to find vertical whitespace rectangles with a high aspect ratio. Among those, the system selects those rectangles that are adjacent to character-sized components on the left and the right side. These whitespace rectangles represent column boundaries with very high probability. In a post-processing step, we eliminate geometrically implausible column boundaries. Columns separators found in this way are shown in Figure 3.

**Text Line Modeling** Text-line finding matches a geometrically precise text line model (Figure 4) against the bounding boxes of the source document; each text line is constrained not to cross any column boundary.[8] Search is currently carried out by a memory-intensive best-first branch-and-bound method; to conserve memory, it will be altered to perform a depth-first search. Each text line is found independent of every other text line, so the orientations of individual text lines can vary across the page (it is, however, possible to constrain text lines to all share the same orientation). The method is specific to Latin script, but the approach generalizes to other scripts.

**Reading Order Determination** The output of column finding and constrained text line matching is a collection of text line segments; it remains putting them in reading order. Reading order is determined by considering pairs of text lines. For certain pairs of text lines, reading order can be determined unambiguously. These pairwise reading order constraints are then extended to a total order by topological sorting.

**Performance** The RAST-based layout analysis summarized above computes exact maximum likelihood solutions to the whitespace and constrained text line matching problems under a robust Gaussian error model; of course, although this model is plausible, it represents a simplification of actual page statistics. When applied to standard ground truthed databases of document images with Manhattan layouts, it yields high performance compared to other standard methods[11–18] (Figure 5).

#### 3.2.3 Other Approaches to Layout Analysis

Although the RAST-based approach to layout analysis described above has turned out to be a reliable workhorse, it has a number of limitations. Most importantly, when it fails, it can only be adapted by modifying a small number of parameters. It would be desirable to have a system that is trainable or adaptable to specific layouts. It is also only applicable to Manhattan layouts. Finally, it only represents one of many desirable cost/performance tradeoffs for layout analysis.

To address these issues, we will likely incorporate the implementations of other layout analysis methods (Voronoi-based, XY-cuts, etc.) into the system. In addition, we are currently developing trainable layout analysis methods that permit explicit representation of a wide variety of layout structures and their geometric variation.

### 3.3 Text Line Recognition

Layout analysis transforms the problem of OCR from a 2D problem into a 1D problem by reducing the page to a collection of "text lines", spatially linear arrangements of characters. Each of these text lines is passed to a text line recognizer for the input document's language. There are already two text line recognizers incorporated into OCRopus, and more will be added in the future.

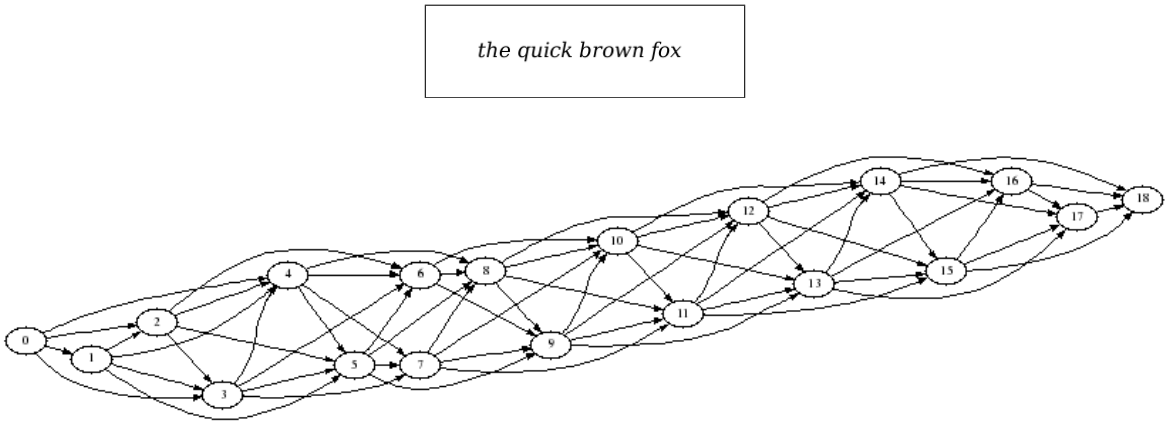


Figure 7. Example of oversegmentation of an input string and representation of the oversegmentation as a finite state transducer (equivalent to a hypothesis graph).

Figure 8. Features used by the MLP-based recognizer include gradients, singular points of the skeleton, the presence of holes, and unary-coded geometric information, such as location relative to the baseline and original aspect ratio and skew prior to skew correction.

### 3.3.1 Tesseract

The Tesseract[20, 21] text line recognizer is based on a mature OCR system developed at Hewlett and Packard (HP) and open sourced recently. Internally, it uses a two-stage shape comparison between prototype character shapes and characters in the input image. Tesseract integrates segmentation of the input image into individual characters with their recognition, using backtracking in case a subsequent state determines that the segmentation is geometrically or linguistically implausible.

Tesseract's character recognition error rates still do not quite achieve the same performance as current commercial systems, but are getting closer, as Tesseract is still under active development. Tesseract is now capable of recognizing many variants of the Latin script, and will likely soon be able to handle other alphabetic languages.

Tesseract does not attempt to estimate character likelihoods, posterior probabilities, or probabilistic segmentation costs, but does return "match scores". Tesseract also does not compute a complete segmentation graph for the input. Both of these factors limit the ability to use Tesseract with the statistical natural language models used by OCRopus.

### 3.3.2 MLP-Based Recognition

A second text line recognizer integrated into OCRopus uses multi-layer perceptrons (MLPs) for character recognition. It proceeds in several steps, first attempting oversegmentation of the input string, then recognizing each potential character hypothesis, and finally expressing both the recognition results and the geometric relationships between character hypotheses as a graph structure.

Oversegmentation is achieved using a dynamic programming algorithm[?, ?] (Figure 6), which also permits kerned and italic character pairs to be segmented. The dynamic programming algorithm identifies potential cut

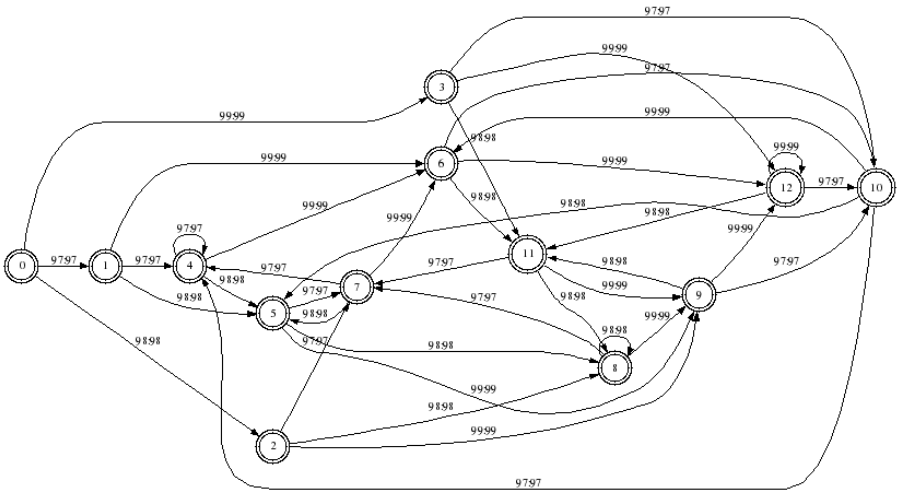


Figure 9. An two letter bigram language model represented as a probabilistic finite state transducer.

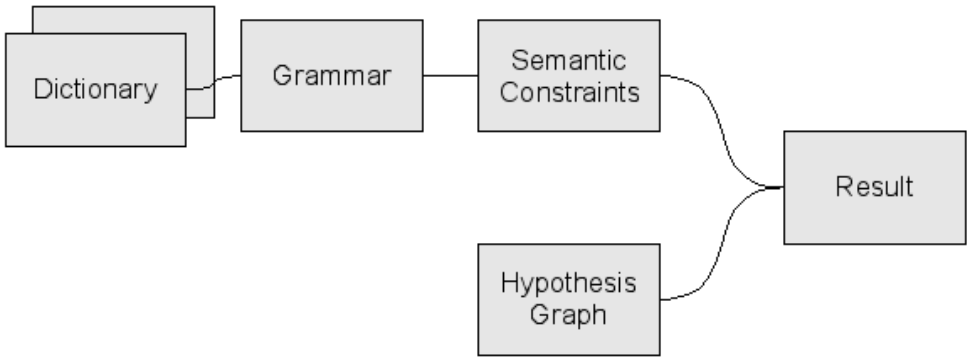


Figure 10. Language models based on finite state transducers can be composed modularly from dictionaries, $n$-grams, grammatical patterns, and semantic patterns. This allows OCRopus to be retargeted and adapted quickly to new document types and languages.

paths between characters. Pairs of nearby cut paths are then considered to be the left and right boundaries of character hypotheses; some of these character hypotheses will correspond to actual characters, while others represent mis-segmented characters. The adjacency relationships between character hypotheses are encoded in a hypothesis graph (Figure 7). For each character subimage, the corresponding image features are computed, determining both the probability that it represents a valid character, and, assuming that it is a valid character, the posterior probability for each class. Features used by the system currently includes gradients, singular points of the skeleton, the presence of holes, and unary-coded geometric information, such as location relative to the baseline and original aspect ratio and skew prior to skew correction.

This combination of oversegmentation, hypothesis graph construction, and later best path identification using dynamic programming has become fairly common in handwriting and OCR systems (although some systems prefer to use backtracking methods or other control structures instead). In particular for text line recognition, the system performs either *maximum a-posterior* (MAP) recognition, or recognition based on full Bayesian posterior probability estimates. The statistical foundations of the system were formulated for speech recognition in[5] and applied to handwriting recognition in[4] (Figure 2).

The MLP currently does not perform as well as Tesseract, either in terms of error rates or speed; however, the network has not been trained extensively, and there is considerable potential for performance improvements.

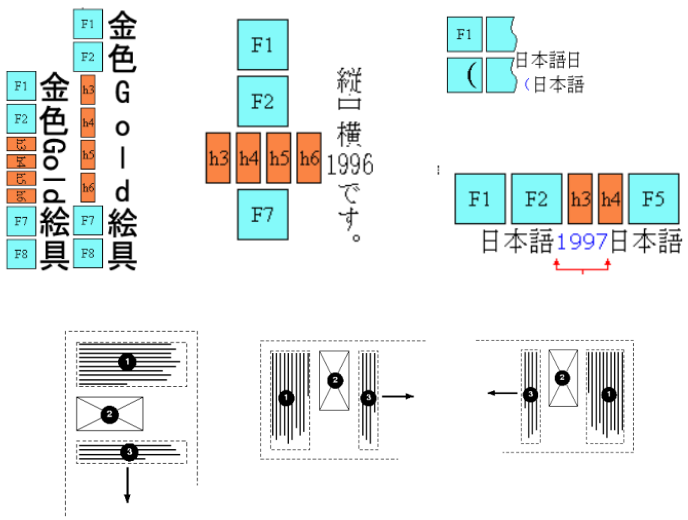


Figure 11. Examples of typographic phenomena that have standard representations in HTML and XHTML–and hence hOCR–but that are not well handled by many other OCR output formats. (Examples taken from W3C.[2])

### 3.3.3 Other Text Line Recognizers

We have also developed a number of other recognizers that we will incorporate into future versions of OCRopus. These include an HMM-based recognizer, suitable for recognizing small font sizes, and an alternative shape-based recognizer, based on prior work on shape-based character recognition.[22]

## 3.4 Language Modeling

The third major component of OCRopus is statistical language modeling. Examples of statistical language models are dictionaries, character-level and word-level $n$-grams, and stochastic grammars. Statistical language models associate probabilities with strings; their function in an OCR system is to resolve ambiguous or missing characters to their most likely interpretation.

Since its alpha release, OCRopus includes the open source OpenFST library[23,24] as the basis of its statistical language modeling tools. OpenFST represents statistical language models as `weighted finite state transducers`. Weighted finite state transducers are a generalization of both hidden Markov models and finite state transducers; they can also be thought of as a form of "weighted regular expression" with the ability to perform limited substitutions. By representing language models as finite state transducers, OCRopus separates language modeling from recognition; that is, complex language models can be prepared off-line using a rich set of language modeling tools, compiled into a weighted finite state transducer representation, and then used as a language model within OCRopus, without OCRopus having to contain any explicit code for complex language modeling tasks.

Weighted finite state transducers can also be used for a variety of other important tasks in OCR:

- expression of character set transformations
- representation of ligature rules and probabilities
- robust information extraction on noisy OCR output
- robust information retrieval on noisy OCR output
- direct statistical machine translation on noisy OCR output

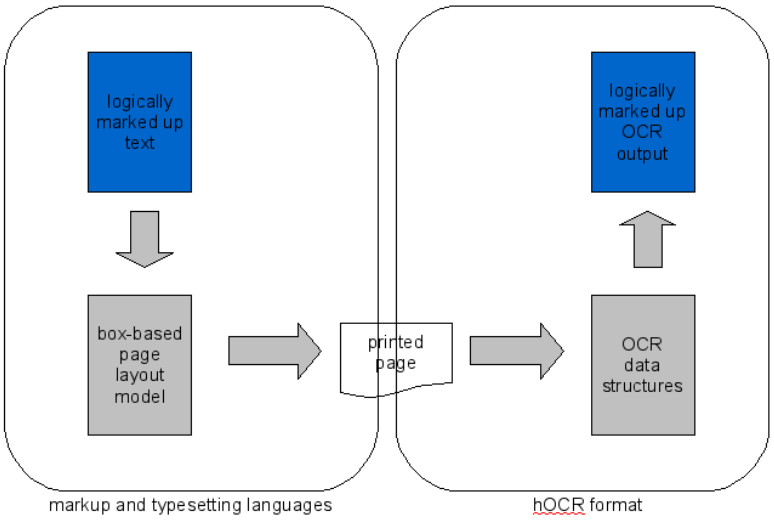


Figure 12. The hOCR format[25] is based on a typesetting model of document generation: documents are generated by flowing logically marked up text into page models consisting of boxes; the hOCR format achieves OCR engine independence by representing these typesetting concepts directly.

A further important aspect of weighted finite state transducers is that they can be used in a modular fashion and can be composed. For example, a general-purpose finite state approximation to English grammar can be combined with domain-specific dictionaries and proper names (Figure 10).

### 3.5 hOCR Output

After the three major processing steps–layout analysis, text line recognition, and statistical language modeling, the system needs to generate a final result, suitable for further processing, reading, editing, indexing, and other applications. Over the years, a wide variety of OCR output formats have been created[26–29] (e.g., XDOC, DAFS, Abbyy-XML). Many of those formats were designed with specific OCR systems and specific script families and languages in mind.

Since OCRopus aims to be an OCR system for all the world's major languages and scripts, none of the existing OCR output formats were suitable. For example, their use of language and layout algorithm-specific notions like "word" and "block" would not mesh well with the multi-language and multi-algorithm approach of OCRopus. Furthermore, none of the existing OCR output formats managed to represent the wide variety of typographic phenomena (reading order, ruby, spacing, line division, etc.) found in different languages and scripts.

OCRopus therefore takes a different approach from existing output formats; instead of starting with an OCR-specific format and adding features for representing typographic phenomena, OCRopus starts with a format that has markup for all major scripts and languages (see Figure 13), namely HTML, and then adds a small number of tags for representing OCR-specific information. The resulting format is called *hOCR*[25] ("HTML-OCR"). Furthermore, the OCR-specific information itself is defined not in terms of algorithmic aspects of specific layout engines, but is based on a two-level typesetting model, with the first level representing logical markup, and the second one representing page layout in terms of boxes and floats (Figure 12). At a third level, the system also permits the encoding of OCR engine specific information, permitting the use of hOCR as a drop-in replacement for other OCR formats within existing workflows.

hOCR documents are standards-compliant HTML or XHTML documents; they can be viewed, edited, and indexed using existing HTML tools. Furthermore, although the format encodes both the original logical and physical layout of the source document (as determined by the OCR system), it is possible to choose a different presentation for the actual HTML. This way, the same hOCR document can support convenient on-screen reading

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN">

<html>
<head>
  <meta name="generator" content="HTML Tidy for Mac OS X (vers 1st December 2004), see www.w3.org">
  <meta name='ocr-id' content='OCRopus Revision: 312'>
  <meta name='ocr-recognized' content='lines text'>
  <meta name='DC.creator' content='Lewis Carroll'>
  <meta name='DC.title' content='Alice in Wonderland'>
  <meta name='DC.publisher' content='Macmillan'>

  <title>Alice in Wonderland</title>
</head>

<body>
  <div class='ocr_page' title='file ../data-ocr-test/alice_1.png'>
    <h3><span class='ocr_line' title='bbox 467 525 1386 588'>1 Down the Rabbit-Hole</span></h3>

    <p class='ocr_par'>
    <span class='ocr_line' title='bbox 461 648 2077 707'>Alice was beginning to get very tired of sitt
    <span class='ocr_line' title='bbox 461 708 2079 766'>and of having nothing to do: once or twice sh
    <span class='ocr_line' title='bbox 460 770 2078 826'>sister was reading, but it had no pictures or
    <span class='ocr_line' title='bbox 459 829 1906 884'>the use of a book,' thought Alice `without pi
    </p>

    <p class='ocr_par'>
    <span class='ocr_line' title='bbox 533 888 2077 946'>So she was considering in her own mind (as we
    <span class='ocr_line' title='bbox 459 947 2079 1007'>day made her feel very sleepy and stupid), w
    <span class='ocr_line' title='bbox 459 1008 2075 1067'>daisy-chain would be worth the trouble of g
    <span class='ocr_line' title='bbox 458 1069 1819 1127'>when suddenly a White Rabbit with pink eyes
    </p>
```

Figure 13. A collection of examples of different typographic phenomena (writing direction, reading order, spacing, etc.) that have standard HTML/CSS representations–and hence are easily represented in hOCR–but are not handled in general by previous OCR output formats.

in a browser or e-book, yet permitting the reader to switch dynamically (using Javascript) to the original layout. Furthermore, the original geometric information remains associated with the text even if presented in logical form.

A sample hOCR document is shown in Figure 13. Note that hOCR has very few required elements, making it easy to retrofit existing OCR engines to output results in hOCR format. hOCR documents are required to indicate, however, which features the OCR engine is capable of generating, so that a user of the output format can determine whether the absence of, say, footnotes in the output is due to their absence from the input document, or due to the inability of the OCR engine to generate this markup. Several document analysis ground truth collections also exist already in hOCR format.

## 4. SOFTWARE ENGINEERING

OCRopus is a large and growing C++ program. The development and debugging of large C++ programs can be difficult, due to the lack of runtime error checking, unsafe type system, and automatic resource management in C++. To keep the project on track, we are adopting several proven C++ development strategies:

- **coding conventions:** OCRopus has a set of coding conventions that keep the source code not only visually consistent, but more importantly set standards for how resource management and errors are to be handled
- **minimizing coupling and interfaces:** OCRopus only permits a small set of data types in its interfaces (scalars, arrays, language models), disallows between-module sharing of global variables, and has only a small number of OCR-specific interfaces.
- **testing:** OCRopus has a large and growing set of unit test, regression tests, and performance evaluation tools

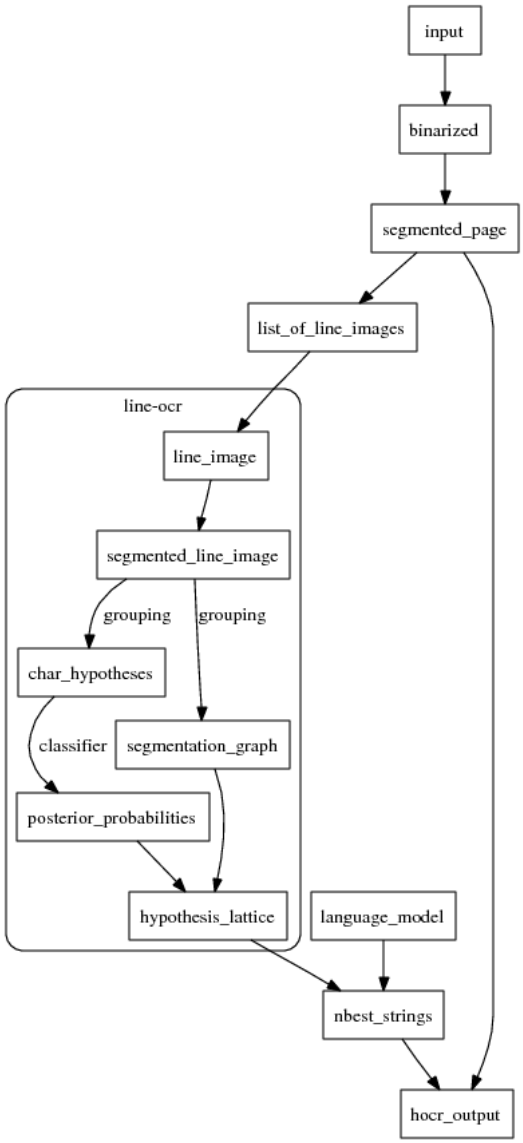


Figure 14. A detailed representation of the intermediate processing steps and representations used by the OCRopus OCR system. Note that all intermediate representations are either images, collections of images, or probabilistic finite state transducers. The final output of the system is in hOCR format–standard HTML4 and CSS3 with DIV and SPAN tags encoding OCR-specific information.

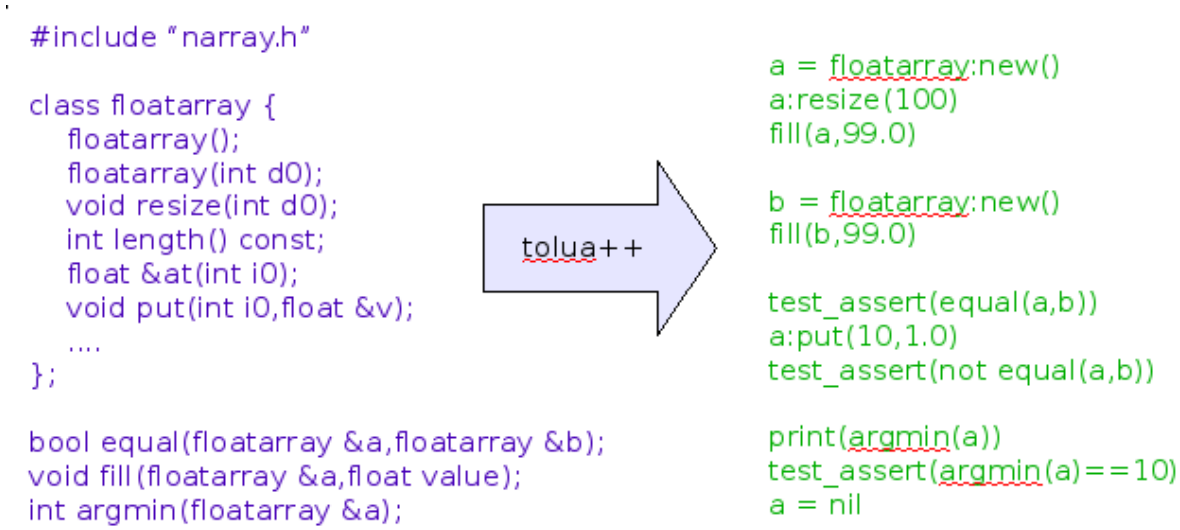


Figure 15. On the left is a C++ class definition, on the right is an example of a test case written in the Lua language using automatically generated bindings for the class on the left.

- **error checking and assertions:** OCRopus uses numerous checks for internal consistency and the validity of arguments and results; these checks come in three levels (expensive, normal, cheap) and can be enabled or disabled selectively to check their performance impact. Production code should ordinarily be run with normal checks enabled.
- **source code metrics and checkers:** Along with OCRopus comes a set of scripts that identifies some coding convention violations, as well as code that is overly complex or violates other metrics.

A further key goal of OCRopus software development is to enable domain experts to express and incorporate their domain knowledge into OCRopus. In order to achieve that, OCRopus is programmable not just in C++, but also in the Lua scripting language.[30] Lua is open source under a liberal license and has a long history of usage in commercial computer gaming; as a consequence, it is very light weight (approx. 100 kbytes), efficient, and easy to learn. Lua has no external dependencies on the file system. Within OCRopus, Lua is not only used for end-user configuration and scripting, but also for unit testing, training, and language modeling. Bindings between C++ and Lua are achieved using an automated tool (tolua++), which also automates garbage collection and exception handling for C++ objects. An example of C++ code and its Lua bindings is shown in Figure 15.

## 5. DISCUSSION

OCRopus is an on-going, active open-source project and has just (October 2007) had its alpha release. OCRopus may already be useful in some applications: according to our benchmarks (Figure 16), it is currently the best available open source OCR system for English, when using the combination of RAST-based layout analysis with the Tesseract text line recognizer.

However, much work remains to be done. In particular, for the beta release in the spring of 2008, we are considering the following features:

- speed improvements in the RAST layout analysis
- incorporation of the HMM-based and shape-based recognizers into OCRopus
- improvements to the MLP-based recognizer
- development of a new A* and beam search decoder, to complement the current OpenFST-based brute force search

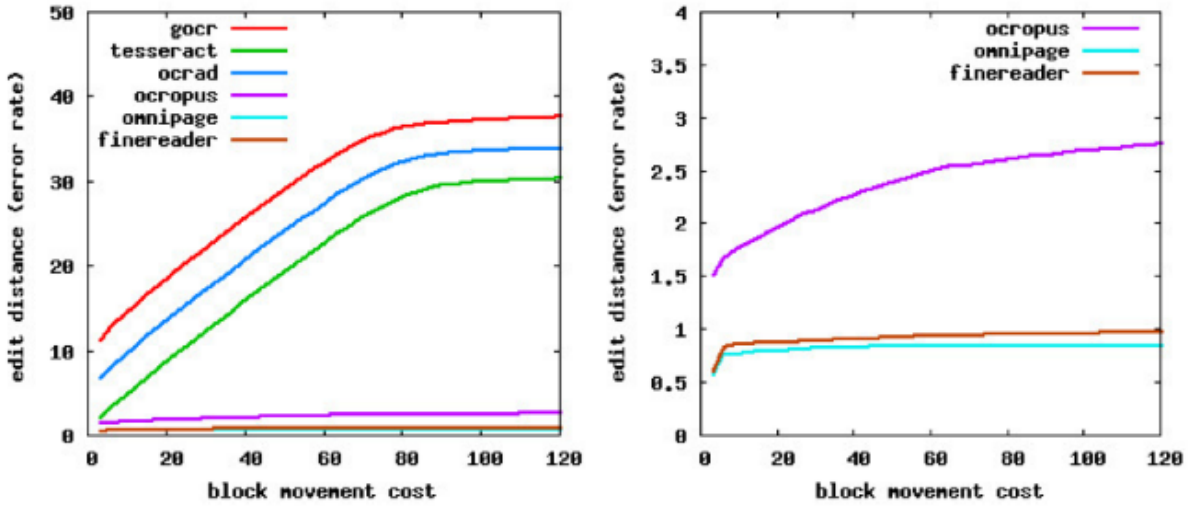


Figure 16. Evaluation of the OCR and layout analysis performance of the OCRopus system on representative pages selected from the University of Washington Database 3 (UW3). The performance measure is string edit distance with unit costs for single character errors and parameterized costs for block moves. Performance near the left end of the graph indicates character-level accuracy, while performance towards the right indicates layout analysis errors. The OCRopus version is from February 2007. Systems compared are the open source gOCR and OCRad systems, and the commercial Omnipage and Abbyy Finereader systems. Tesseract is evaluated once as a stand-alone system, and once as a text line recognizer within the OCRopus system.

- a usable language model (dictionary plus character $n$-gram) for OCR of English text
- improved training tools
- extensive testing and overall speed and error rate improvements
- the ability to create a usable version of OCRopus with no external library dependencies

Longer term, there are important needs in the areas of character exception processing, mathematical formula recognition, language and script identification and font and style identification. Furthermore, from a research perspective, we are focusing on two areas:

- on-the-fly adaptation of character shape, layout, and language models to entire documents
- statistically sound semi-supervised training for OCR applications
- trainable and adaptive document layout models

Overall, our hope is that OCRopus will become a widely adopted standard for research in OCR, as well as the basis for many open source and commercial applications, in areas ranging from digital libraries to affordable reading machines for the visually impaired. Potential contributors can find out more about joining the project at `www.ocropus.org`; we are particularly interested in new text line recognizers for a variety of scripts and language modeling tools for different languages. The source code for the system can be downloaded at the same address.

## ACKNOWLEDGMENTS

## REFERENCES

1. "HTML/XHTML Specifications." http://www.w3.org/MarkUp, 2006.
2. "CSS: Under Construction." http://www.w3.org/Style/CSS/current-work, 2006.

3. F. Shafait, D. Keysers, and T. M. Breuel, "Pixel-accurate representation and evaluation of page segmentation in document images," in *18th Int. Conf. on Pattern Recognition*, pp. 872–875, (Hong Kong, China), Aug. 2006.
4. T. M. Breuel, "Design and implementation of a system for the recognition of handwritten responses on us census forms," in *Proc. of DAS-94: International Association for Pattern Recognition Workshop on Document Analysis Systems*, pp. 109–134, (Kaiserslautern), 1994.
5. H. C. Leung, I. L. Hetherington, and V. W. Zue, "Speech Recognition using Stochastic Explicit-Segment Modeling," in *EUROSPEECH 91. 2nd European Conference on Speech Communication and Technology Proceedings*, Instituto Int. Comunicazioni, (Genova, Italy), 1991.
6. T. M. Breuel, "Binary morphology and related operations on run-length representations," in *3rd International Conference on Computer Vision Theory and Applications, 22 - 25 January, 2008, Funchal, Madeira - Portugal*, 2008. accepted for publication.
7. F. Shafait, D. Keysers, and T. M. Breuel, "Efficient implementation of local adaptive thresholding techniques using integral images," in *Document Recognition and Retrieval XV*, (San Jose, USA), Jan. 2008. Accepted for publication.
8. T. M. Breuel, "Robust least square baseline finding using a branch and bound algorithm," in *Document Recognition and Retrieval VIII, SPIE, San Jose*, pp. 20–27, 2002.
9. F. Shafait, J. van Beusekom, D. Keysers, and T. M. Breuel, "Page frame detection for marginal noise removal from scanned documents," in *15th Scandinavian Conference on Image Analysis*, pp. 651–660, (Aalborg, Denmark), June 2007.
10. D. Keysers, F. Shafait, and T. M. Breuel, "Document image zone classification - a simple high-performance approach," in *2nd Int. Conf. on Computer Vision Theory and Applications*, pp. 44–51, (Barcelona, Spain), Mar. 2007.
11. F. Shafait, D. Keysers, and T. M. Breuel, "Performance comparison of six algorithms for page segmentation," in *7th IAPR Workshop on Document Analysis Systems*, pp. 368–379, (Nelson, New Zealand), Feb. 2006.
12. T. M. Breuel, "High performance document layout analysis," in *Symposium on Document Image Understanding Technology, Greenbelt, MD*, 2003.
13. S. Mao, A. Rosenfeld, and T. Kanungo, "Document structure analysis algorithms: a literature survey," *Proc. SPIE Electronic Imaging* **5010**, pp. 197–207, Jan. 2003.
14. H. S. Baird, "Background structure in document images," *Bunke, H. and Wang, P. S. P. and Baird, H. S. (Eds.), Document Image Analysis, World Scientific, Singapore* , pp. 17–34, 1994.
15. G. Nagy and S. Seth, "Hierarchical representation of optically scanned documents," in *Proc. of the 7th Intl. Conf. on Pattern Recognition*, pp. 347–349, (Montreal, Canada), 1984.
16. G. Nagy, S. Seth, and M. Viswanathan, "A prototype document image analysis system for technical journals," *Computer* **7**(25), pp. 10–22, 1992.
17. K. Y. Wong, R. G. Casey, and F. M. Wahl, "Document analysis system," *IBM Journal of Research and Development* **26**(6), pp. 647–656, 1982.
18. K. Kise, A. Sato, and M. Iwata, "Segmentation of page images using the area Voronoi diagram," *Computer Vision and Image Understanding* **70**, pp. 370–382, June 1998.
19. T. Breuel, "Segmentation of handprinted letter strings using a dynamic programming algorithm," in *Proceedings of Sixth International Conference on Document Analysis and Recognition*, pp. 821–6, 2001.
20. R. Smith, "The Tesseract open source ocr system." http://code.google.com/p/tesseract-ocr/.
21. R. Smith, "An overview of the Tesseract OCR engine. ," in *Int. Conf. on Document Analysis and Recognition (ICDAR), Curitiba, Brazil*, 2007.
22. T. Breuel, "Recognition of handprinted digits using optimal bounded error matching," in *Proceedings of the 2nd International Conference on Document Analysis and Recognition (ICDAR '93)*, p. 493ff, 1993.
23. "Openfst library." http://www.openfst.org/, 2007.
24. C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri, "Openfst: a general and efficient weighted finite-state transducer library," in *Proceedings of the Ninth International Conference on Implementation and Application of Automata, (CIAA 2007)*, pp. 11–23, 2007.

25. T. M. Breuel, "The hOCR Microformat for Workflow and Results," in *Int. Conf. on Document Analysis and Recognition (ICDAR), Curitiba, Brazil*, 2007.
26. B. A. Yanikoglu and L. Vincent, "Ground-truthing and benchmarking document page segmentation," in *Proc. 3rd Intl. Conf. on Document Analysis and Recognition*, pp. 601–604, (Montreal, Canada), Aug. 1995.
27. RAF Technologies, Inc. (Redmond, WA), "Illuminator user's manual." http://documents.cfar.umd.edu/resources/source/illuminator.html, 1995.
28. D. S. Connelley and B. Paddock, "Xdoc data format: Technical specification." Xerox Imaging Systems part no. 00-07571-00, 2000.
29. I. Guyon, R. M. Haralick, J. J. Hull, and I. T. Phillips, "Data sets for OCR and document image understanding research," in *Handbook of character recognition and document image analysis*, H. Bunke and P. Wang, eds., pp. 779–799, World Scientific, Singapore, 1997.
30. "The lua language." http://www.lua.org/, 2007.